# Szakdolgozat

**Mikó Sándor**

**Debrecen**
**2010**

**Debreceni Egyetem**
**Informatikai Kar**

# Adatmodell alapú felhasználói felületek Javával

**Témavezető:**
**Kósa Márk**
**egyetemi tanársegéd**

**Készítette:**
**Mikó Sándor**
**Programtervező**
**Informatikus**

**Debrecen**
**2010**

# TARTALOMJEGYZÉK

1. Bevezetés ..... 2

2. A Java nyelv ..... 3
2.1. Története ..... 3
2.2. Jellemzői ..... 3

3. A Swing ..... 4
3.1. A Swing főbb jellemzői ..... 4
3.1.1. A Swing architektúrája ..... 5
3.1.2. Eseménykezelés a Swing számára ..... 7
3.2. A Swing komponensei ..... 13
3.2.1. Szövegfelületek ..... 13
3.2.2. Gombfelületek ..... 16
3.2.3. Listák ..... 18
3.2.4. Csúszka ..... 19
3.2.5. Folyamatjelző ..... 20
3.2.6. Menük ..... 21
3.2.7. Táblázat ..... 22
3.2.8. Kész komponensek ..... 25

4. Program bemutatása ..... 28
4.1. Program telepítése ..... 29
4.2. Program használata ..... 30
4.3. Gyakorlati jelentősége ..... 39

5. Összegzés ..... 40

6. Irodalom ..... 41

# 1. BEVEZETÉS

Szakdolgozatom témája a Java - val rendelkező, adatmodell alapú felhasználói felületek. Írásomban azt vizsgálom, hogy a Java Swinggel hogyan lehet egyszerű és egyben gyors felhasználói felületet programozni és hogy ez alkalmas - e egy komolyabb, gyakran használt, üzleti logikájú program megvalósításához.

Mivel programtervező informatikusnak tanulok, ezért mindig is a legnagyobb vágyam az volt, hogy egy olyan önálló programot tervezzek, amelynek gyakorlati haszna is van. Egyik ismerősöm autósiskolát vezet és mesélte, hogy nem megfelelő a programjuk, mindig baj van vele. Ezen felbátorodva határoztam el, hogy készítek az autósiskolának egy átláthatóbb, egyszerűbb tanuló nyilvántartó rendszert, egy olyan programot, amely még inkább személyre szabott és nem tartalmaz felesleges extra funkciókat, ezzel is elősegítve a hatékonyságot.

Dolgozatom három fő részből áll. Az első részben röviden bemutatom a Java nyelvet, annak történetét és jellemzőit.
A második részben megvizsgálom a Swing – et, annak főbb jellemzőit, komponenseit. Azok közül is részletesebben tárgyalom a szövegfelületeket, a gombfelületeket, a listákat, a csúszkát, a folyamatjelzőt, a menüket, a táblázatokat és a kész komponenseket.
Az utolsó részben pedig ismertetem az elkészült programom jellemzőit, telepítését, használatát és gyakorlati jelentőségét.

Dolgozatomban arra törekszem, hogy a lényeges momentumokat ábrákkal, táblázatokkal és képekkel is szemléltessem, hogy a hétköznapi, a számítógépekhez kevésbé értő emberek is könnyedén kiigazodjanak benne.

# 2. A JAVA

## 2.1. Története

A Java története 1991-ben kezdődött a „Green” projekttel. A Sun MicroSystem egy csoportja Patrick Naughton és James Gosling vezetésével egy mini, számítógépes nyelvet tervezett, amely kisméretű, hordozható és hatékony kóddal rendelkezik. Bár tervüket sikerült megvalósítani, sokáig senkit sem érdekeltek ennek a nyelvnek a lehetőségei. 1994-ben jött el az igazi áttörés, amikor is a Java fejlesztői megalkották a Hot Java böngészőt (Java futtató környezettel), amely segítségével a honlapok már mozgó ábrákat, animációkat és videófilmeket is képesek voltak megjeleníteni. A sikerét az hozta meg, hogy egy virtuális gépre közbenső kódot generál. Innen ered a JVM (Java Virtual Machine) elnevezés. Ezután ez a kód minden gépen használható megfelelő interpreter segítségével. A Java a C++ célirányos leegyszerűsítése.

## 2.2. Jellemzői

A Java nyelv legfontosabb jellemzői, pozitívumai:

**Egyszerű** - A C++ nyelv továbbfejlesztésének nevezhető. Egy olyan egyszerűsített változata, amely objektumorientált nézetben avn. Emiatt egyszerűbb és átláthatóbb benne programozni.

**Objektumorientált** - Az *objektumorientált* (OO) paradigma középpontjában a programozási nyelvek absztrakciós szintjének növelése áll. Ezáltal egyszerűbbé, könnyebbé válik a modellezés, a valós világ jobban leírható, a valós problémák hatékonyabban oldhatók meg. A világot vagy problémát egy modellként írjuk le, melynek elemei az osztályok. Az osztályok rendelkeznek attribútumokkal és metódusokkal. Ebből az osztályokból példányosítás során keletkeznek az objektumok. Egy objektumnak mindig van egy rá jellemző állapota, amit beállító metódusokkal tudunk befolyásolni. Az osztályok egymásból származtathatók, így jön létre az öröklődés és jelenik meg az újrafelhasználhatóság. Az OO nyelveknek két nagy csoportja van, a tiszta és a hibrid OO nyelvek. A java „majdnem” tiszta OO nyelv, mert OO paradigma mentén épül fel és egy osztályhierarchiát tartalmaz.

**Biztonságos –** a tervezők igyekeztek a nyelvet úgy megalkotni, hogy a hibák minél hamarabb kiderüljenek, és ez által kevésbé viselkedjenek abnormális módon. A Java - ban ezen kívül szabályozhatjuk az objektumok, metódusok, attribútumok láthatóságát és hozzáférhetőségét is.

**Gépfüggetlen** – Egy, a számítógéptől független bájtkódot (.class file) állít elő a Java fordító. Ebből a JVM (Java Virtual Machine) garantálja, hogy a bájtkód minden platformon ugyanúgy viselkedjen.

**Többszálú** – Egy időben több szál, folyamat futhat. Egy processzoros rendszerek esetén ez azt jelentheti, hogy próbálja a rendszer „igazságosan" elosztani. A több processzoros rendszerekben ez eloszlik és így a futási idő radikálisan csökkenhet.

Legnagyobb hátránya a lassúsága, de jó hír hogy a mai otthoni átlagos számítógépeken ez már nem okoz gondot.

# 3. A SWING

## 3.1. A Swing főbb jellemzői

A Java két API - t biztosít a grafikus felhasználói felület (Graphical User Interface, GUI) programozására, ezek közül a Swing a modernebb, fiatalabb és manapság használatos könyvtár. A Swing előtti időkben az egyszerűbb felületeket az AWT – vel (abstract window toolkit) fejlesztették. Legnagyobb hátránya emellett, hogy már nem fejlesztik tovább és, hogy a felület kialakításához csak korlátozott mértékű komponensek állnak a rendelkezésünkre. Ennek ellenére ma is fontos az AWT, hiszen az osztályait a Swing - gel létrehozott felületek is mai napig használják. Egy jó Swing programozónak feltétlenül kell ismernie a Java API dokumentációját, mert a Swing kiterjedése olyan nagy, amit csak ott tudunk nyomon követni.

**AWT** (Abstract Window Toolkit, absztrakt ablak eszköztár)
A gép saját natív elemeit használja, ezek a nehézsúlyú elemek (heavyweight). Így platformfüggő, hogy mit is eredményez egy ugyanazon kód. Az AWT a java.awt csomagban kapott helyet.

A Java fejlesztői rájöttek, nem kedvező, hogy a különböző platformokon másképp jelennek meg a különböző elemek. Ezért kibővítette az AWT gyűjteményét pehelysúlyú (lightweight) komponensekkel. A pehelysúlyú elemek már nem az operációs rendszer gépi kódú elemeit használják, hanem a Swing készíti azokat el, így az lassabb, mint elődje. A mai fejlett hardverekkel már ez sem jelent problémát. Mivel a Swing az AWT kibővítése, ezért gyakran használja annak elemeit. A Swing osztályok a java.swing csomagban találhatók és a benne lévő eszközök közös jellemzője, hogy J betűvel kezdődnek.

A Swing osztályrendszere nem olyan könnyen átlátható, mint az AWT - é. Azonban az AWT elavult, így szükség van az újításra. A Swing rengeteg dolgot tartalmaz, ami megkönnyíti a programozó munkáját. Többek között a címkéken használható ikonok, vagy a grafikus elemek megjelenítésénél a kettős pufferelési technika.

### 3.1.1. A Swing architektúrája

A Swing - API architektúrája az MVC- architektúrájának modelljére támaszkodik. Ez az jelenti, hogy minden komponensnek (például egy parancsgomb vagy egy szövegmező) van egy Modellje, legalább egy Controller - je és egy View – ja. A Modell az adatok megfelelő feldolgozásáért (üzleti logika, az adatok meghatározott feldolgozása egy cél érdekében), a View megjelenítésért (amit a képernyőn látunk, az eredmény) és a Controller pedig a bemenetért felelős (tipikusan a beviteli eszközök tartoznak ide, vezérlők). Többségében jellemző, hogy több mint egy Controller tartozik egy View-hoz és több View létezik egy Modell esetén.

A Component és Container osztályok képezik az üzemrendszer WindowManagger[1] interfészét. Minden üzemrendszerhez különböző Wrapperek szükségesek, amelyek a Java világát egy, az üzemrendszeren menetképes környezetbe burkolják. A Container osztály több Component elemet tartalmaz, amiket egy gyűjtőbe foglal és kezel. A JComponent absztrakt osztály a legfontosabb osztály, mivel ez tartalmazza a Swing működéséhez szükséges elemeket. A komponensek fajtájától függően a fejlesztő saját adatmodellt készíthet el.

A Swing egyik lényeges különlegessége, a Pluggable Look and Feel a Controller és View delegáthoz való kötegelésén alapul. A Pluggable Look and Feel azt jelenti, hogy lehetséges a komponensek kinézetét és viselkedését futási időben megváltoztatni.

A Java a különböző platformok futási időkörnyezete. Minden platform felhasználója hozzászokik, az azon futó programok meghatározott kinézetéhez és viselkedéséhez. Mivel a felhasználók hozzászoktak ehhez és az idegenül ható programokat inkább nem szívesen használják, a Swing azt a lehetőséget nyújtja, hogy a grafikus felület kinézetét és viselkedését a mindenkori üzemrendszerhez illesszék. Így a Swing alkalmazások a felhasználó szokásos környezetébe beleilleszkednek. Szerencsére erre nem kell programsort írni. Ha egy program elindul, a futási időkörnyezet először kiválasztja a Swing alkalmazások számára tipikus felületet.

Ha másik felületet szeretnénk alkalmazni futási időben, akkor ehhez az UIManager setLookAndFeel () osztálymódszerét kell használnunk. Egy példa a használatára: UIManager.setLookAndFeel(stílus);

Ahol a stílus a következő táblázat valamelyike lehet:

| | |
|---|---|
| Metal (Standard) | `javax.swing.plaf.metal.MetalLookAndFeel` |
| Synth | `javax.swing.plaf.synth.SynthLookAndFeel` |
| Multi | `javax.swing.plaf.multi.MultiLookAndFeel` |
| CDE/Motif | `com.sun.java.swing.plaf.motif.MotifLookAndFeel` |
| Windows (XP) | `com.sun.java.swing.plaf.windows.WindowsLookAndFeel` |
| Windows Classic | `com.sun.java.swing.plaf.windows.WindowsClassicLookAndFeel` |
| GTK+ | `com.sun.java.swing.plaf.gtk.GTKLookAndFeel` |

[1] Az üzemrendszer része arra, hogy a grafikus elemeket, mint az ablakot, ábrázolja és kezelje.

Nem minden Look and Feel áll rendelkezésre minden üzemrendszerben. GTK+ csak az unix-linux platformokban jeleníthető meg. Windows alatt csak az alapértelmezett Metal, Windows (XP) és Windows Classic. A Metal minden Java program számára egy minta, amit alapvetően támogat.

A Look and Feel minták tartós rögzítésére is lehetőség van a swing.defaultlaf paraméter segítségével. Ennek egy, az előző táblázatban feltüntetett stílust értékként adva történik.

### 3.1.2. Eseménykezelés a Swing számára

Ha a modell adatváltozását tekintjük egy eseménynek, akkor a modell az esemény megtörténte következtében kommunikációt nyit meg a view - val. A modell egy hírt küld a view - nak azért, hogy az adatváltozás eseményét megossza vele.

Az esemény feldolgozásban mindig legalább három objektum vesz részt:

- Az eseményforrás (Event-Source) egy olyan objektum, amely eseményhíreket generál, mint például egy interfész.
- Egy eseményhír (Event-Object) lehívja az eseményt és ehhez passzoló szöveginformációkat szállít.
- az esemény elfogadó (Event-Listener) egy olyan objektum, amely az eseményekre reagálni szeretne. Egy esemény elfogadó egy olyan tárgy, ami például egy interfész működéséről értesít.

Az MVC minta modelljének és view-jának a példája ezt könnyen beláthatóvá teszi. A modell a hír forrása, a view pedig az esemény címzettje. Először egy kommunikációs kapcsolatba lépnek, miután a view a futási időben a modellnél regisztráltra magát, hogy az adatváltozásról informálva legyen. Ennek a laza összekapcsolásnak az előnye abban áll, hogy a forrás és az címzett egymástól függetlenül fejleszthető és megváltoztatható. Például a modell a view-tól függetlenül fejleszthető.

A források és a címzett felfejtésének előnyei:

- Függetlenebb fejlődést tesz lehetővé a forrásoktól és a címzettektől.
- Takarékos kommunikáció! Csak az olyan tárgyakat informálják az eseményről, ami a forrásnál regisztrálva van.
- Gyenge összekapcsolás a forrás és a címzett között.

A forrás és címzett között egy olyan laza kapcsolat van, amelyben a hírek a fellépő eseményekre cserélődnek ki. Egy ilyen hírt eseményhírnek is nevezünk. Az eseményhírek amellett az információ mellett, hogy egy meghatározott esemény fellépett, szöveg információkat is hordoznak az eseményről. Ha például az egér gombját megnyomjuk, akkor az üzemrendszerből egy eseményhírt küldenek a programhoz, ahol az egér mutatója aktuálisan tartózkodik. Az eseményhír szöveginformációként az egér mutatójának koordinátáit tartalmazza a képernyőn. A program a koordináták alapján tudja meghatározni, hogy a gomb lenyomására mit kell reagálni.

Az üzemrendszer és az esemény továbbadásához szolgáló program közötti kommunikáció aszinkron a normális program folyamattal. Ha fellép egy esemény, akkor egy alkalmas helyen a normális programfolyamat megszakad és eseménykezelés valósul meg. A Windows alatt az eseményeket egy olyan pufferben végzik el, amit az üzemrendszer tölt meg. A program a pufferből elveszi az eseményeket és feldolgozza azokat. Más rendszerek hasonló mechanizmusokat alkalmaznak, hogy az eseményeket aszinkronon a programokhoz továbbvezessék.

Az esemény feldolgozás a következő folyamattal írható le:

- Először a címzettnek a forrásnál regisztrálnia kell magát, hogy egy belépő eseménynél automatikusan értesítsék. Minden forrásnál egyidejűleg több címzett is regisztrálhat.
- Miután az esemény belépett, a forrás egy eseményhírt küld minden regisztrált címzettnek.

Az esemény feldolgozáson a hírek aszinkron feldolgozását értik. Ezeket a híreket többnyire a felhasználó váltja ki. Például egy gomb lenyomásával vagy egy ablak minimalizálásával.

A Swingben a címzettre, a forrásokra és az eseményekre speciális osztályok és interfészek vannak, azért hogy a programozást leegyszerűsítsék.

**Eseményhierarchia**

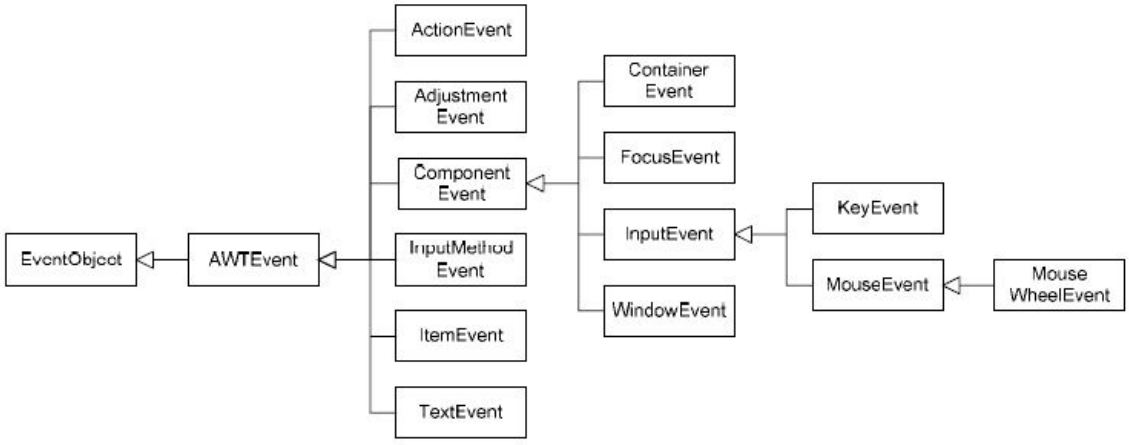
A java.util csomagból származó EventObject osztály minden esemény bázisosztálya. A csomag alapján máris fel lehet ismerni, hogy az Eventekről való kommunikáció nem csak egyedül a felületi programozásra alkalmas, hanem más területeken is értelmes lehet. Az EventObject osztály a getSource() metódust tartalmazza, ami az eseményforrásnak egy referenciát ad vissza, ami a hírt kiváltotta:
public Object getSource();

A grafikus felületek számára jelenlévő események szülőosztálya a java.awt csomagból az AWTEvent osztály. Ez az EventObjectből van levezetve és az AWT és Swing minden eseményének a szülőosztálya. A legtöbb eseményosztály a java.swing.event és javax.awt.event csomagokban található.

Egy eseményforrás gyakran különböző eseményeket válthat ki és oszthat fel. Ennél egy forrás több eseményét gyakran egy osztályba kapcsolják, hogy az osztályhierarchiát átláthassák. A java.awt.event.MouseEvent osztályban például miden eseményt megvalósítanak, amit az egér válthat ki. Azért, hogy egy eseményhír elfogadásánál a pontos eseményt megtalálják, minden eseménynek van egy saját ID - eseménye. Az események ID - it speciális eseményosztályokban határozzák meg. Az AWTEvent osztályban a getID() módszer valósul meg. Ez az esemény egyértelmű esemény ID - jét adja vissza:

public int getID();

Saját események megvalósítása is lehetséges. Ennél arra kell vigyázni, hogy az események ID - jei az AWTEvent.RE-SERVED_ID_MAX érték felül helyezkedjenek el, mert a Swing és AWT alkalmazására már minden érték fenn van tartva. A következő képen egy áttekintés következik a legfontosabb eseményosztályokról:




Az AWTEvent eseményhierarchiája

Az események a „Low-Level” és a „High-Level” eseményeibe vannak beosztva. A „Low-Level” eseményeket a felhasználó váltja ki, amelyeket a rendszeren keresztül elfogadnak, és Java alkalmazáshoz továbbvezetnek. Az események forrása az üzemrendszer (illetve a felhasználó). Jó példa erre a billentyűzet egyik gombjának a lenyomása. A „High-Level” eseményeket közvetlenül a Java programon belül hozzák létre. Ilyen események forrásai például a Swing komponensei.

**Az események megfigyelése**

Azért, hogy egy Object bizonyos eseményeket fogadni tudjon, meg kell valósítani egy eseményfajtához passzoló Listener interfészt és regisztrálnia kell magát az eseményforrásnál. Az esemény forrás egy esemény fellépésekor aktiválja a regisztrált objektumoknál a Listener interfész megfelelő módszereit és továbbadja az esemény tárgyát.

Minden esemény számára van egy vagy több Listener interfész, ami minden eseménytípus számára tartalmaz egy megfelelő módszert. Például a MouseListener osztály megfelel a MouseEvenet eseménynek. A MouseListenerben például a mousePressed() és a mouse-Released() módszerek deklarálva vannak, amelyeket a címzett objektumnak kell megvalósítani. A módszerek aktiválásánál csak egy egyedi paramétert, az eseményobjektumot adják át. A Controller esemény feldolgozását, a Java-ban alkalmas Event-Listener valósítja

meg. Minden Listenert a java.util.EventListener vezeti le és a java.awt.event csomagban vannak.

**Eseményforrások**

Minden tetszés szerinti objektum lehet egy eseményforrás. A felületi programozásnál mindenekelőtt a komponensek és a containerek, tehát például az ablak, a gombok, a menük vagy a görgetősávok (Scrollbars), amelyek az eseményhíreket elküldik. Azért, hogy az eseményforrás egy eseményt elküldjön az esemény címzettjének, előbb az eseményforrásnál regisztrálnia kell magát. Regisztráció nélkül a forrás titkos marad és a címzett így nem kap információt a fellépő eseményről.

Eljárási mód az eseménykezelés megvalósításában:

- Az eseményforrást elfogadni,
- Az esemény címzettjét meghatározni,
- A címzettet a forrásnál regisztrálni.

A regisztráció addXYZListener() minta szerinti függvényekkel következik, ahol az XYZ az esemény fajtája, például addMouseListener() azért, hogy az egéreseményekelfogadására egy objektumot regisztráljon. Ezekhez a módszerekhez szokásosan egy olyan objektum referenciáját adják át, amely a megfelelő Listener interfészt megvalósítja. Általánosan a komponensek több módszert kínálnak, amelyekkel címzettként egy tárgyat különböző eseményekre regisztrálhatnak.
Egy példa:

```
public class ButtonFrame extends JFrame implements ActionListener
{
        public ButtonFrame()
        {
                JButton okButton = new JButton ("OK");
                okButton.addActionListener (this);
        }
                public void actionPerformed(ActionEvent ae)
        {
        //Ide kell az esemény kiváltására történő kódot megírni.
        }
}
```

A fenti forráskódban az ok.Button.addActionListener (this) aktiválásával futási időben egy objektumot adnak át, ami az ActionListener interfészt megvalósítja. Az interfész előírja az acitonPerformed() függvény megvalósítását. A JButton típus forrásánál egy gombról van szó. Az actionPerformed() függvényt a forrás aktiválja, ha a felhasználó megnyomta a gombot.

Lehetséges tetszés szerint sok címzettet egy eseményforrásnál regisztrálni. Egyébként nem lehet megjósolni, hogy a források milyen sorrendben informálják a címzetteket egy esemény fellépésekor.

**Eseménykezelés és konkurencia**

Az eseménykezelés aszinkron a normális program folyammal. Egyébként vigyázni kell arra, hogy csak az eseménykezelésbe való beugrás aszinkron. Az esemény tulajdonképpeni kezelése azonos folyamatszálban történik, amelyben a program szokásszerűen fut. Ebből a kijelentésből a következő következtetéseket kell levonni:

- Ha valamilyen okból kifolyólag az a metódus, amelyik az eseménykezelést vezeti, blokkolva van, akkor az összes program megáll.
- Hogyha több mint egy címzett jelentkezik a forrásnál, akkor a következő címzettek már nem lesznek informálva az eseményről.
- Ha az eseménykezelés több mint 40 ms - ot igényel, akkor a felhasználó lassulást fog érzékelni. Ezért az események kezelésénél vigyázni kell arra, hogy a felhasznált idő lehetőleg rövid legyen. Ha több időt használunk fel, akkor ez nem lesz tragikus, amíg a kezelést ritkán aktiváljuk. Ha a kezelés aktiválása mégis gyakran fordul elő, akkor meg kell változtatni ezt.

Mivel a Java multiszálasíthatásra képes, mint minden más osztálykönyvtárban, a Swing - nél is felvetődik a kérdés, hogy hogyan érintkezik a felületen több szál, versengő hozzáférésével. Az AWT tervezésénél nagy hangsúlyt fektettek a multiszálasítás biztonságára. Megállapították, hogy különböző szálak a felületen tudnak együtt dolgozni anélkül, hogy egymást akadályoznák vagy blokkolnák. Ez többek között az AWT felületek hiányzó sebességének egyik oka. A Swing osztálykönyvtárnál néhány kivétellel lemondanak a szálbiztonságról.

Tehát el kellene kerülni, hogy több szál egyidejűleg a Swing felületén dolgozzon, azért hogy például az adatok aktualizálását a modellben végrehajtsák. Tehát ha az a szükségszerűség áll fenn, hogy a felületen több szállal kell dolgozni, akkor ezt a SwingUtilities osztály invokeLater() és invokeAndWait () metódusaival végzik el. Ott a szál kivezetése a felületi EventQueue[2]-be van sorolva.

A SwingWorker osztályával a Java 6 óta egy további, egy már széles körben elterjedt lehetőség jött arra, hogy a hosszantartó munkameneteket a háttérfolyamatokban kiürítsék. A SwingWorker osztály addig létezik, ameddig a Swing, bár eddig csak a JDK-n kívül szeparált API-kban állt rendelkezésre. Az alkalmazás hasonló, mint egy szálé, bár a SwingWorker instanciája a folyamatadatokat a grafikus felület szálára adja vissza. A JProgressBar komponens bemutatásánál a SwingWorker osztályt használják, hogy a háttérfolyamatok folyamatfejlődését lekérdezzék, majd a felületen ábrázolják.

## 3.2. A Swing komponensei

### 3.2.1. Szövegfelületek

#### JLabel

A JLabel osztály egy statikus szöveg, amit a felhasználó nem tud megváltoztatni. Szokás címkének nevezni. A JLabelnek ikonkép is átadható ilyenkor statikus képként viselkedik. A getText() és setText() metódusokkal kérhető le és állítható be.

```
JLabel label = new JLabel ("Név");
```

Például:

Név:

[2] Minden eseményt, amely egy Swing felületre hatást gyakorol – például az egérgomb vagy a billentyűzetgomb -, egy EventQueue-be (eseményvárakozási sor) sorolják be, aztán szekvenciálisan kidolgozzák.

### JTextField

Egy JTextField egy szöveges mező, amit futási időben változtatható, egyfajta beviteli mezőként működik a felhasználótól a számítógép felé.

JTextField textFeld = new JTextField ("");

Így egy üres szövegmezőt kapunk. Ha szeretnénk megadni kezdeti értékét, akkor a konstruktor idézőjelei közé írva megtehetjük.

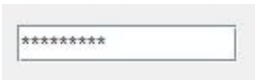
Példa a JLabel és a JTextField együttes használatára:


### JPasswordField

A JPasswordField egy kibővítése a JTextField osztálynak. Ez is egy beviteli mező, ami megváltoztatható, de itt a beírt adatok elfedésbe kerülnek, jelszó mező.

Itt is adhatunk deklaráláskor kezdő értéket a mezőnknek. Az alapértelmezés szerint a megjelenített szöveget tömör pontokkal fedi el. Ezt megváltoztathatjuk például '*'– jelre:

jPasswordField1.setEchoChar('*');




### JFormattedTextField

Ez is a JTextField osztály kiegészített változata, ami egy Format objektum példánnyal testre szabhatóvá teszi a megjelenített szöveget. Ha számformázást szeretnénk, akkor átadható neki például egy NumberFormat példány, amellyel számformázást valósíthatunk meg, de akár a szöveg háttérszínét is egyszerűen manipulálhatjuk:

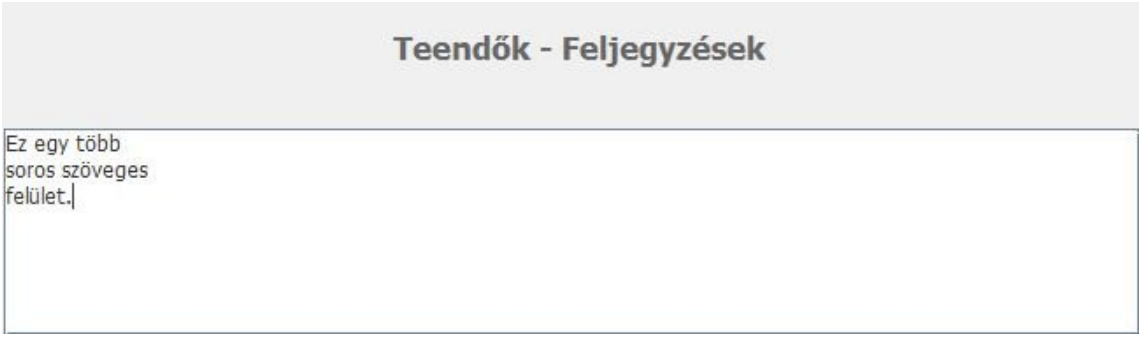
JFormattedTextField formattedTF = new JFormattedTextField(new Double(1900.24));

formattedTF .setForeground(Color.red);

## JTextArea

Ugyancsak a JTextFieldhez köthető, azzal a különbséggel, hogy ezzel több soros szöveges beviteli felületet tudunk létrehozni. Alkalmas egy hosszabb szöveg vagy akár egy dokumentum megjelenítésére. A görgetősáv automatikusan, de csak szükség esetén jelenik meg.

JTextArea textArea = new JTextArea ("Ez egy több\nsoros szöveges\nfelület.");




## JEditorPane

A JEditorPane típus alkalmas a HTML tartalmak és egyszerűbb szövegek megjelenítésére. A HTML alapú megjelenítés előnye, hogy aki jártas a webes programozásban úgy, mint a weblapoknál itt is a HTML szabályaihoz igazodva formázhatja a megjelenést, amit egy böngészőben megjelenítve, rendezett HTML oldalt kap.

JEditorPane editor = new JEditorPane();

editor.setContentType ("text/html;charset=UTF16");

editor.setText ("<html><h1>Cím</h1>HTML-Text</html>");

### 3.2.2. Gombfelületek

#### JButton

Egy, a felhasználó által billentyűzettel vagy egérrel létrehozott esemény kiváltására alkalmas. Így a JButton lehet egy ”Mentés” vagy ”Mégse” gomb, ami mögött a választástól függően különböző esemény futhat le.

JButton gomb= new JButton ("Mentés");

JButton gomb= new JButton ("Mégse");


#### Icons

Az ikonok olyan, a JButton-nak átadható komponensek, mellyel grafikus felületet adhatunk a gombunknak. Így elérhetővé válik, hogy a háttértáron lévő kép állományokból testreszabott gombot hozzunk létre, amit vizuálisan kiemel, akárcsak az előző példában is látható a Mentés és a Mégse feliratok előtt.

ImageIcon ikon = new ImageIcon ("Mentés.jpg");

JButton button = new JButton ("Mentés", ikon);

#### JToggleButton

A JToggleButton egy egyszerű eszköz, aminek a segítségével választási lehetőségeket ajánlhatunk fel a felhasználó számára.

```
JToggleButton kilépés = new JToggleButton ("Kilépés");
JToggleButton mégse = new JToggleButton ("Mégse");
```

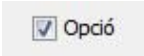
A fenti sor segítségével létrehozhatunk gombokat a megfelelő szöveggel és utána ezeket egy gyűjtőcsoportba rendezhetjük:

```
ButtonGroup csoport = new ButtonGroup();
csoport.add(kilépés);
csoport.add(mégse);
```

A csoportosítás lényege, hogy több gombfelület esetén, amik egy csoportba vannak, csak az egyik kiválasztását engedélyezzük.

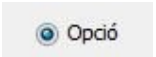
### JCheckBox

Opciók kiválasztását teszi lehetővé, a JToggleButton kibővítése. Az ezeket kijelölő négyzeteket egérrel ki tudjuk választani, amit például egy pipa vagy egy kereszt jelez. Konstruktorának második paraméterben átadható egy logikai érték, mellyel engedélyezhetjük egy adott opció előre kijelölését. Alapértelmezésben ez az érték ”false”.




### JRadioButton

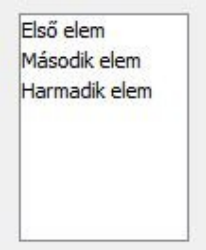
Mindenben megegyezik az előbb látott JCheckBox-al, azzal a különbséggel, hogy míg ott négyzetben történt a kijelölés, itt körökben. Mindkettőnél alkalmazható a már említett, egy csoportba való besorolásra való ButtonGroup osztály.



### 3.2.3. Listák

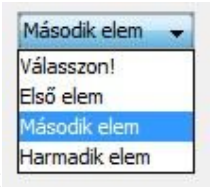
#### JList

A lista egyszerre való objektumok megjelenítésére és kiválasztására. A létrehozáskor egy Object tömböt kell átadnunk, amivel a lista elemeit feltöltjük. Object [] elemek= { "Első elem", "Második elem", "Harmadik elem" };




#### JComboBox

A JComboBox egy szöges mező és egy lista kombinációja. Az első elemre kattintás után lenyílik egy lista. Ebből a listából, ha kiválasztunk egy elemet, a szöveges mező tartalma megváltozik. Egy String osztályt kell itt is átadnunk az elemek létrehozásához. Be lehet állítani, hogy melyik listaelem legyen alapméretezetten kiválasztva a szövegmezőben a setSelectedIndex(int Index) metódussal lehet. Az indexek számozása 0-tól kezdődik. Szerkeszthetővé is tehetjük, amivel elérhetjük, hogy a listában nem szereplő adatot adjunk meg a szöveges mezőnek. Egy nem szerkeszthető, 4 elemű, JCombobox




:

#### JSpinner

A JSpinner-t el lehet képzeli egy listaként, amiben egy gomb megnyomásával haladhatunk felfelé illetve lefelé. Ez a lista lehet szöveges, például a hónapokból készített lista vagy állhat 0 – 100 közötti számokból, amit könnyen változtathatunk. A JSpinner deklarálása után a

setModel() metódussal tudjuk testre szabni, aminek egy SpinnerModelt-t kell átadni. A következő példában egy 0-150 double intervallum áll rendelkezésünkre, amiben 1.5 értékkel lépegethetünk.

jSpinner1.setModel(new javax.swing.SpinnerNumberModel(0.0d, 0.0d, 150.0d, 1.5d)); Az első paraméter a kezdeti érték, a második és a harmadik a minimum és a maximum, az utolsó pedig a lépésköz. A paraméterek után álló ”d” betű jelzi a tizedes ábrázoláshoz szükséges double típust.


### 3.2.4. Csúszka

#### JSlider

A JSlider segítségével egy értéktartományból grafikusan választhatjuk ki a nekünk megfelelő értéket. Itt két fajta grafikus jelölés van, a láthatóságot segítő kisebb és nagyobb részbehúzás. Főbb metódusai, amivel beállíthatjuk a JSlider objektumot:

jSlider1.setMajorTickSpacing(500); A nagyobbik részbehúzást mutató jelölés beállítása.
jSlider1.setMinimum(2000); A minimum érték.
jSlider1.setMaximum(2000); A maximális érték.
jSlider1.setMinorTickSpacing(100); Másodlagos részbehúzó jelölés.
jSlider1.setPaintLabels(true); A nagyobb behúzásokhoz tartozó számok megjelenítése.
jSlider1.setPaintTicks(true); A nagyobb és a köztük lévő kisebb behúzások megjelenítése.
jSlider1.setValue(1000); A csúszka kezdeti értékhez tartozó pozíció beállítása.

Ezen kívül tetszés szerint elhelyezhető vízszintesen vagy függőlegesen az előző paraméterekkel megadott komponens kinézete.

### 3.2.5. Folyamatjelző

#### JProgressBar

A JProgressBar segítségével időigényes folyamatok feldolgozottsági állapotát tudjuk szemléltetni.

Fontosabb metódusai:
setStringPainted(true) - A futás állapotát számmal kiírja %-ban.
setValue() – Kezdeti érték.
setMinimum() és setMaximum - Intervallum megadása.
setIndeterminate() -

Ahhoz, hogy változást érjünk el, implementálni kell egy SwingWorker osztályt és annak metódusait.

Egy példa:

```
SwingWorker sw = new SwingWorker() {
   protected Object doInBackground() throws Exception {
      for (int i = min; i <= max; i++) { // Intervallum és a lépésköz megadása.
      setProgress(i); //Érték szerinti kirajzolás.
        try {
       Thread.sleep(70); // 70 ms várakozás két lépés között.
        } catch (Exception ex) {
        }
     }
   return null; // Itt visszatérhetünk tényleges értékkel is, amit a StringWorker get()
metódusával lekérdezhetünk.
  }
};
```

A doInBackground() mellett létrehozhatunk egy done() metódust is, ami 100% elérésekor megváltoztatja a szöveget valami értelmes üzenetre, például ”Kész van!” feliratra.

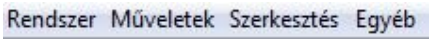
Íme, az eredmény egy lehetséges állapota:


És egy lehetséges végállapot:


### 3.2.6. Menük

#### JMenuBar

A Swingben a JMenuBar osztállyal lehet menülécet létrehozni, így a programok funkciót könnyen csoportosíthatjuk, amivel megkönnyítjük a felhasználó tájékozódását. Erre épül a többi menüfajta, ehhez lehet hozzáadni további részeket az add() metódus segítségével. Az egér segítségével tudjuk kibontani az elemeit.



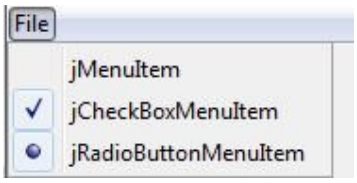

#### JMenu

A JMenuBar része. Létrehozáskor a konstruktornak átadhatunk szöveget és így az adott névvel létrejön. Az előző példában négy darab JMenu látható, melyek nevei sorban Rendszer, Műveletek, Szerkesztés és Egyéb menürészek. Ezek gyűjtő csoportok, melyekre kattintva egy lenyíló menülistát kapunk. Azért, hogy az egyes elemeket jól el tudjuk különíteni, alkalmazhatjuk az addSeparator() metódust.

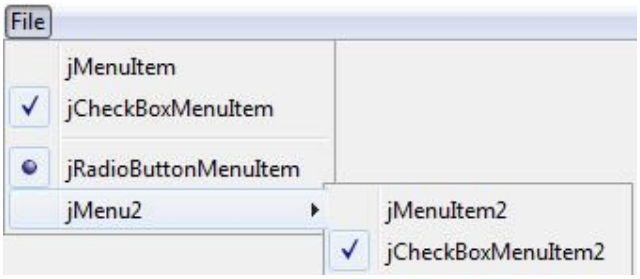
#### JMenuItem

A JMenu-hez tartoznak, a lenyíló menülista konkrét elemei. Létezik az egyszerű JMenuItem és két leszármazottja: JCheckBoxMenuItem és a JRadioMenuItem. Ez utóbbi kettőnél lényeges, hogy mint a gomboknál, ezek is hasonló tulajdonságokkal rendelkeznek. Egyrészt

kinézetre, másrészt pedig ButtonGroup segítségével itt is manipulálhatjuk a kijelölés típusát. Ezen kívül az egyes menüelemekhez megadhatunk ikonokat és gyorsbillentyűket is.




Ezt a kialakított hierarchiát lehet tovább bonyolítani almenükkel. Ezt úgy lehet elérni, hogy a már meglévő JMenu példányunkhoz, egy újabb JMenu objektumot adunk hozzá az add() metódus segítségével. Utána hasonlóan adhatunk hozzá különböző JMenuItem fajtákat.




### JPopupMenu

A Pop - Up menüt vagy másképpen a kontextus menüt a JPopMenu osztály hozza létre. Egy normális menüvel ellentétben, ehhez nem kapcsolnak menülécet és a programozónak magának kell a menü megjelenéséről gondoskodni. Így, hogy nincs fix menüléc, az aktuális felületen a JPopupMenu-t bárhol elő tudjuk hozni a jobb egérgomb lenyomásával. Ezen kívül megegyezik az alap menüvel, az elemek típusai, felépítése és kinézete.

## 3.2.7. Táblázat

### JTable

A Swing API segítséget biztosít az adatok szép és rendezett megjelenítésére. Erre a célra szolgál a táblázat. Egy kicsit több mindent kell a programozónak beállítania, függően attól mennyire akarja testre szabni. Alapvetően két fontos megvalósítás létezik. Az egyik, hogy a konstruktorának egy kétdimenziós adatokat tartalmazó tömböt adunk át, a másik mód, hogy

az adatmodellt maga a programozó adja meg. Így adataink alkalmazásához vagy a TableModel-t vagy a DefaultTable-t kell használnunk.

Utóbbi nagyobb testreszabhatóságot és több metódust ad a kezünkbe, ezért is választottam az én a programomhoz a DefaultTable modellt a táblázataim elkészítésekor. Mindkettőnél alkalmazható a getValueAt(int sor, int oszlop) metódus, amivel egy adott pozíciójú elem értékét kérhetjük le. A példakód a JTable példányosítása után így néz ki:

```
((DefaultTableModel) jTable1.getModel()).setRowCount(0);  // ki töröljük az összes sort ha van...
 jTable1.setModel(new javax.swing.table.DefaultTableModel(
 new Object[][]{},
 new String[]{ "", "Név", "Iktató szám","Kategória", "Tanfolyam szám", "Érvényesítve", "Oktató neve" }) {
 Class[] types = new Class[] // Megadjuk az egyes oszlopok típusát. Itt nem csak java.lang.String.class lehet, hanem bármilyen másik java osztály, amit a JTable kezelni tud. (Objectből származik) Pl.: Integer, Long, Boolean, stb.
 {
            java.lang.Integer.class,
            java.lang.String.class,
            java.lang.String.class,
            java.lang.String.class,
            java.lang.String.class,
            java.lang.String.class,
            java.lang.String.class
};
  boolean[] canEdit = new boolean[] // ha módosíthatónak szeretnéd az oszlopokat, akkor true értéket kell megadni.
{
            false,
            false,
            false,
```

```
            false,
            false,
            false,
            false
};
        public Class getColumnClass(int columnIndex) {
            return types[columnIndex];
        }
        public boolean isCellEditable(int rowIndex, int columnIndex) {
            return canEdit[columnIndex];
        }
    });
    DefaultTableModel dtmMy = (DefaultTableModel) jTable1.getModel();
```

Ezzel létre is jött a modellünk. Ezek után egy String tömböt készítünk az adatokkal és végül beszúrjuk a DefaultTableModel példányunkba.: dtmMy.insertRow(sorszamlalo, sor); A sorszámláló jelzi azt, hogy hova szúrjuk be a táblánkba az egy sort reprezentáló String tömböt, ami nulláról indul. A testreszabhatóság szemléltetésére én a táblámban bizonyos oszlopok szélességét átállítottam. Egy TableColum objektum értékének a jTable1.getColumnModel().getColumn(oszlop_száma); - át adtam. Ezek után a létrejött példány különféle metódusaival manipulálhatok. Fontosabb függvényei: setMinWidth() és setMaxWidth() az oszlop minimum és maximum szélességét állítják be, a setResizable() pedig az átméretezhetőséget. Szűrést és rendezést is beállíthatunk adataink számára a Java 6 óta. Egy RowSorter objektumot kell létrehoznunk és megírni, ami alapján történjen a szűrés és rendezés, amit majd a setRowSorter(táblázat) meghívása aktivál. Ezek után már csak egy példával zárnám le a JTable bemutatását. Íme:

| | Név | Iktató szám | Kategória | Tanfolyam szám | Érvényesítve | Oktató neve |
|---|---|---|---|---|---|---|
| 1 | Tanuló1 | 2 | C1+E | 123 | 2010.03.02 | Oktató1 |
| 2 | Ferike | 21 | D+E | 1/2 | 2010.03.01 | Mikeux |
| 3 | Zsoldoskatona | 2010/03/04 | B | 2010/456 | 2010.03.04 | OKtató3 |
| 4 | Valaki1 | 32432 | TR | 43242 | 2010.02.02 | OKtató3 |
| 5 | Tanuló90 | 1 | A1 | 1 | 2000.02.02 | OKtató 9999 |
| 6 | Tanuló91 | 2 | C | 2 | 2000.02.02 | OKtató 9999 |
| 7 | Tanuló93 | 3 | D+E | 3 | 2000.02.02 | OKtató 9999 |
| 8 | Tanuló 94 | 4 | TR | 4 | 2000.02.02 | OKtató 9999 |
| 9 | Tanuló95 | 5 | D1+E | 5 | 2000.02.02 | OKtató 9999 |
| 10 | Tanuló01987 | 5 | B+E | 5 | 2000.02.07 | OKtató 9999 |
| 11 | KisPista | 2010.04.14 | D1+E | 2010/1546 | 2010.04.14 | OKtató 9999 |

### 3.2.8. Kész komponensek

#### JOptionPane

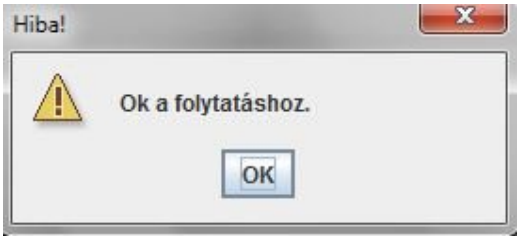
Egy program állapot informálására használják, amivel egyszerűen tájékoztathatjuk a felhasználót.

Fajtái:
- showConfirmDialog Ellenőrző kérdést tesz fel, amire a válasz lehetőségek: Igen/Nem/Mégse
- showInputDialog Egy beviteli mező segítségével, befolyásolhatjuk a futást.
- showMessageDialog Egy információs ablak, amivel értesíthetjük a felhasználót egy aktuális eseményről.
- showOptionDialog Különböző fajtájú lehet. A másik három kombinációja, amit testre szabhatunk.

Főbb paraméterek:

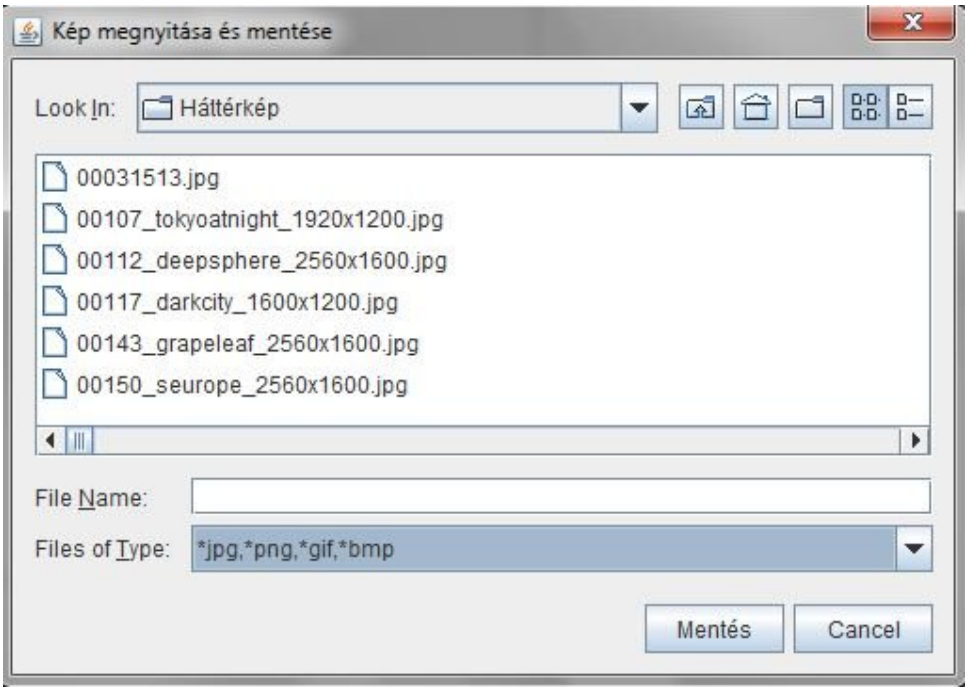
- message A kiírandó üzenet.
- icon A megjelenítendő képet lehet változtatni.
- messageType A megjelenítést befolyásolják. Típusai: ERROR_MESSAGE, INFORMATION_MESSAGE, WARNING_MESSAGE, QUESTION_MESSAGE, PLAIN_MESSAGE
- optionType A gombok számának, fajtájának beállítása: DEFAULT_OPTION, YES_NO_OPTION, YES_NO_CANCEL_OPTION, OK_CANCEL_OPTION

Egy példa a WARNING_MESSAGE alkalmazására:



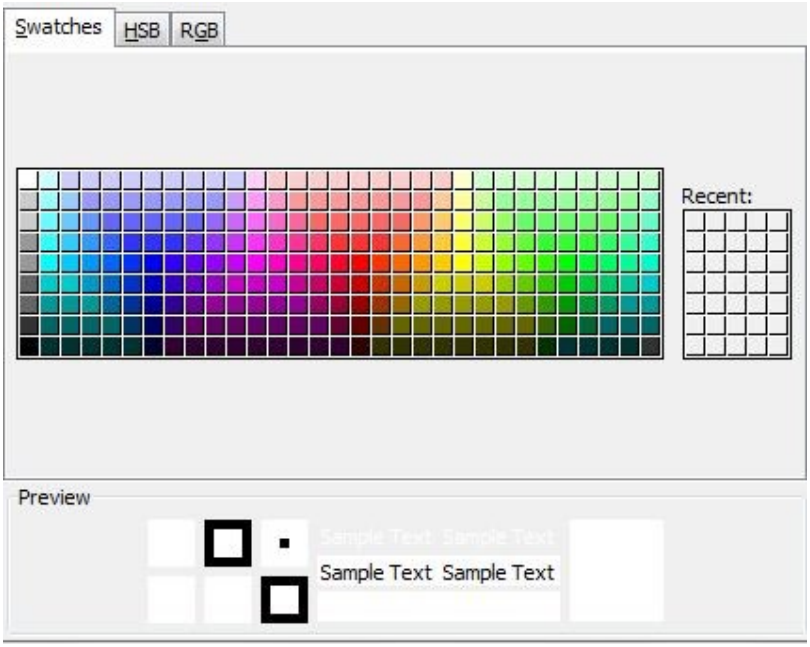
### JFileChosser

Egy fájlválasztó dialógus, fájlok, könyvtárak kiválasztását teszi lehetővé. A megjelenítéshez a showOpenDialog()-ot kell meghívni és máris az aktuális fájlrendszer látható. Az alapméretezett könyvtár beállítása a setCurrentDirectory() - val történik. Sok fajta állapotát lehet vizsgálni, például a APPROVE_OPTION érték jelzi, hogy a felhasználó kiválasztott egy fájlt és a Mentés gombra kattintott. Ezek után a getSelectedFile() metódussal elérhetjük a fájlt, amit feldolgozhatunk, lementhetünk vagy akár meg is jeleníthetünk. Még szerintem lényeges tulajdonság, ha valaki például csak képeket vár kiválasztásra, akkor azt alapméretezetten be tudja állítani. Erre szolgál a setFileFilter() metódus, aminek egy FileFilter osztály példányát kell megadni, ahol testre szabhatjuk, hogy mi legyen alapméretezetten megjelenítve.




### JColorChosser

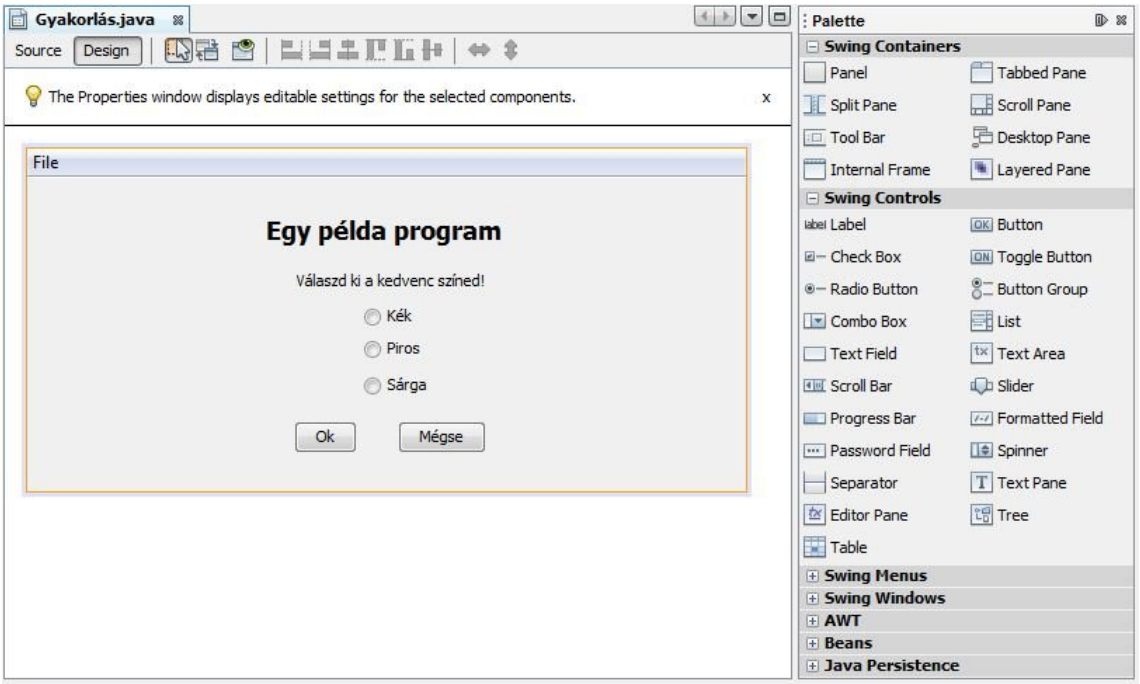
Ez egy színválasztó dialógus, színek kiválasztására szolgál. Megjelenítéshez itt is a showDialog metódust kell meghívnunk. Ezután a felhasználó tetszőleges színt keverhet, akár RGB, HSB számokat is megadhat, majd egy Color objektumot kap visszatérési értékként.




Ezek lennének a Swing komponensek a teljesség igénye nélkül, amik a programomhoz köthetők és a lényegesebbek. Mindegy egyes elem sok-sok metódussal rendelkezik, amikkel testre szabhatjuk őket és csak részben soroltam fel. Az összes megtalálható a Java Swing API-ban. A komponensek gyűjteménye elérhető a következő címen:

http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/package-summary.html

Nagy könnyítés jelenthet egy jó integrált fejlesztői környezetben dolgozni. Én a program elkészítése során a NetBeans IDE - t (Integrated Development Environment) választottam, ami szép és átlátható körülményeket biztosít a programozónak. Egy grafikus ablakba, a Swing komponenseket bele tudjuk húzni és még a futás előtt meg tudjuk nézni annak kinézetét. További előnyei, hogy a behúzott elemeket jobb egérkattintással beállíthatjuk tetszés szerint anélkül, hogy komolyabb programkódokat írnánk. A következő képen a NetBeans fejlesztői környezete látható, aminek legfrissebb verziója letölthető a http://netbeans.org címről.




# 4. PROGRAM BEMUTATÁSA

A program tervezésekor, megalkotásakor az volt a célom, hogy egy olyan programot készítsek az autósiskola számára, amely átláthatóbb és egyszerűbb tanuló nyilvántartó rendszerrel rendelkezik.

A hangsúly azért került az egyszerűségre és az átláthatóságra, mert az iskola eddigi rendszere sok olyan alkalmazást tartalmazott, amelyek feleslegesek voltak a felhasználó számára, és olykor a munkáját is megnehezítették.

Programomat e program alapján írtam meg, de törekedtem arra, hogy még inkább személyre szabott legyen, és hogy a leglényegesebb funkciókat akár a menüből is el lehessen érni.

## 4.1. Program telepítése

Ebben a részben megtudhatjuk, hogy bírható működésre a mellékelt program. Mint látjuk adott egy .jar és egy .sql kiterjesztésű fájl. A .jar tartalmazza a tényleges programot, aminek a futtatásához, ha még nincs feltelepítve a gépünkön, a Java futtató környezete (JRE,Java Runtime Environment) szükséges. Ez letölthető a Java hivatalos honlapjáról: http://www.java.com/en/download/index.jsp
Miután kiválasztottuk a nekünk megfelelő típust és feltelepítettük, máris futtathatóvá válik a .jar fájlunk.

Ezen kívül a program mysql adatbázist használ az adatok tárolásához és kiolvasásához. Telepítenünk kell egy servert a gépünkre. Én személy szerint a XAMPP Litle ingyenes programot használom, aminek a telepítését röviden összefoglalnám. A http://www.apachefriends.org/en/xampp-windows.html oldalon kiválasztva itt is a megfelelő Litle típust választhatunk futatható és csomagolt verzió közül. A csomagolt fájl esetében, miután letöltöttük a számítógépünkre és kicsomagoltuk, a főmappában lévő setup_xampp.bat segítségével konfigurálhatjuk egyszerűen.

1. Akarunk-e a Start menüben és az asztalon indító ikonokat?
2. „y” a folytatáshoz.
3. A XAMPP hordozható legyen e? (portable)
4. Folytatáshoz Enter.
5. Aktuális időzóna feltüntetése.
6. 1-gyel  a vezérlő panelt elindíthatjuk.
7. Utolsó lépésként az Apache és MySql start-al való indítása szükséges és itt lehetséges, hogy engedélyezni kell a tűzfalunkon, az általuk használt portokat.

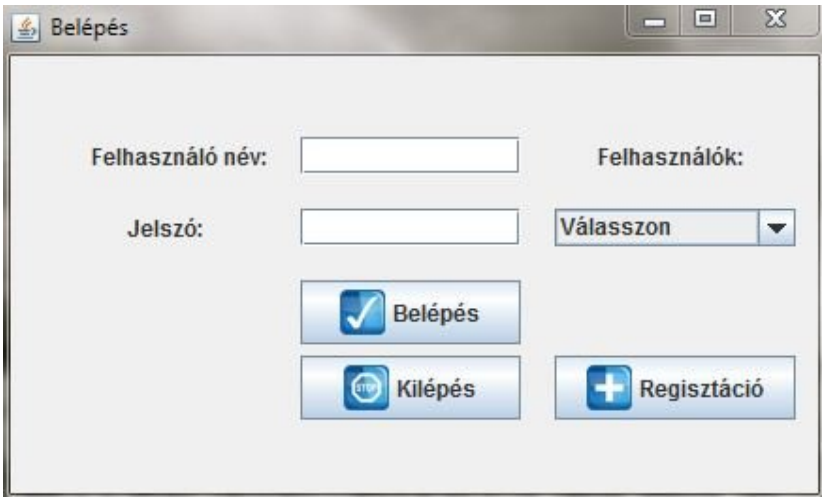
Az sql fájl tartalmazza az adatbázis táblák létrehozásához és inicializálásához szükséges kódot. Ennek futtatásához egy böngészőre van szükségünk. A címsávba beírva a localhost címet már is server főoldalára érkeztünk. Itt a PhpMyAdmin szólgáltatást kell igénybe vennünk, ott importálhatjuk a sql fájlunkat, mely így lefutva már is létrehozta a szükséges táblákat. Még egy felhasználó létrehozása maradt a serverünkön, ami privilégiumok, vagy felhasználók néven találunk meg. Neki megadva a szükséges jogokat a localhost serveren (ajánlott teljes jogosultságot adni) és megadva neki a megfelelő felhasználónevet és jelszót **(depo, depo)**, működőképes lesz a program.

Akinek már van, feltelepített lokális szervere a gépén, annak csak le kell futatni a fájlt és a megadott paraméterekkel egy felhasználót kell létrehoznia. Fontos, hogy a program futása alatt a MySql servernek kell futnia, tehát az előbb elindított Apache-ot leállíthatjuk. Indulhat a program!

Ha a "Nem jött létre adatbázis kapcsolat." üzenetet kapjuk, akkor az adatbázis szerverünk nem fut, vagy az adatbázis táblák nem jöttek létre.

## 4.2. Program használata

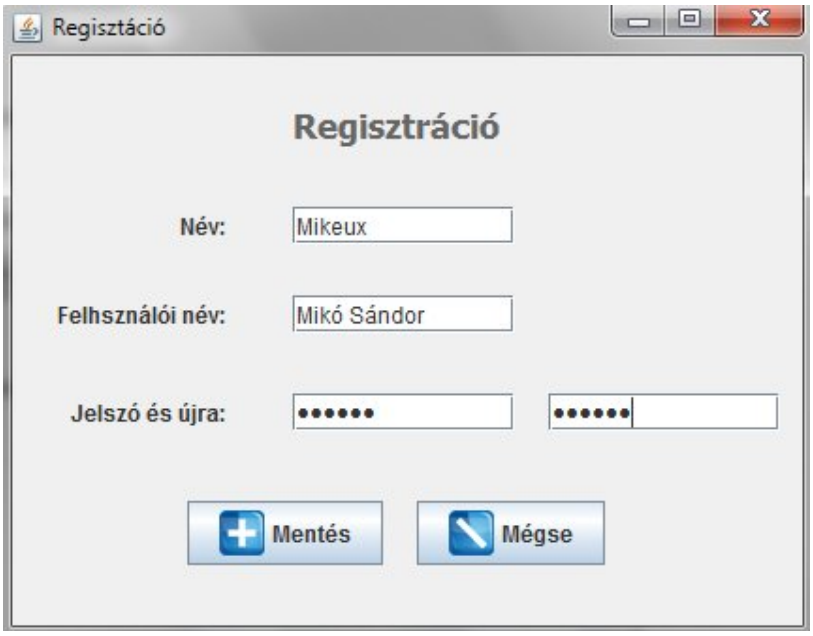
A program a MySql szerver futtatása után a depo.jar fájlal indítható. Helyes beállítások esetén egy bejelentkező képernyő fogad.



## Regisztráció

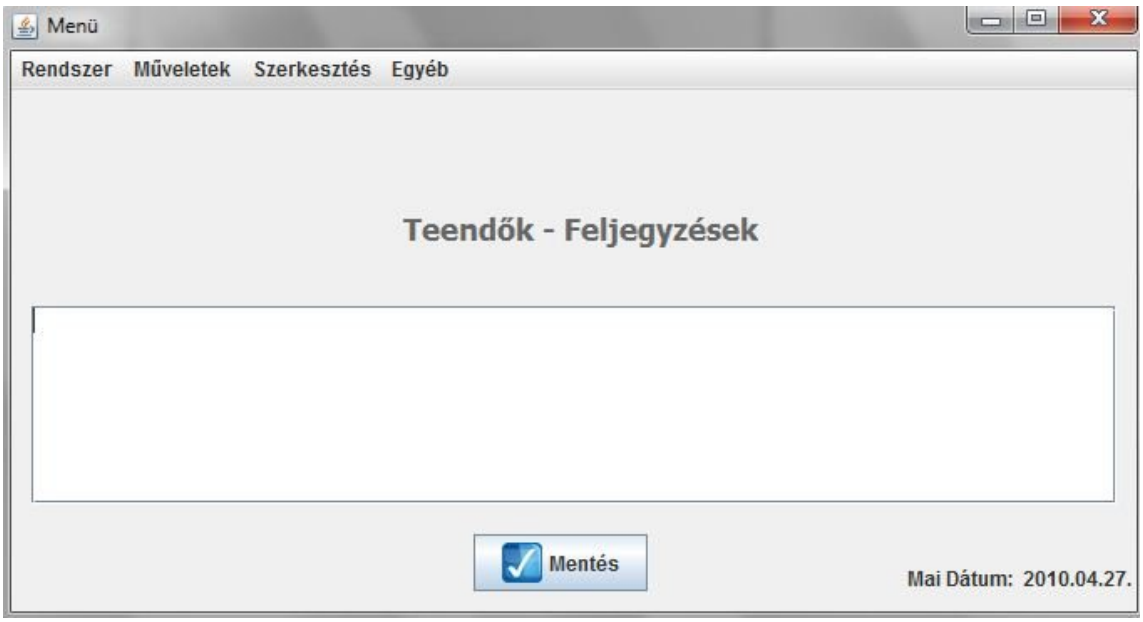
Első futtatáskor még az adatbázisunk üres, így felhasználók sem léteznek. Ezek jelenléte azért is fontos, mert minden bejelentkezéskor naplózva van, hogy mikor, milyen névvel használják a programot. Így nyomon tudjuk követni, akár több felhasználó esetén is, hogy ki, mit csinált, de ez még később részletesebben meg lesz említve. Regisztráció során hozhatunk létre új felhasználókat. Itt adni kell minden mezőnek értéket és a jelszómezőknek is egyeznie kell, ellenkező esetben a program figyelmeztet minket. Általánosan igaz a programra, hogy a hibákat és egyéb visszajelzéseket legtöbbször az aktuális ablak alján, pirossal jeleníti meg.

Ha elkészítettük a felhasználónkat, akkor a bejelentkező képernyőn a felhasználók listából ki tudjuk választani és így automatikusan kitöltésre kerül a felhasználó név és jelszó mezők. Nem maradt más hátra, mint a belépés.




## Főmenü

A legfontosabb része a programnak a bejelentkezés után látható menü, mert innen tudjuk elérni a különböző funkciókat.




A közepén elhelyezkedő nagy felületre tudunk emlékeztetőket, feljegyzéseket írni és elmenteni. Így a következő futás során is megmaradnak ezek az információk. Jobb olalon alul láthatjuk még a mai dátumot.

A következőkben a menük részeit tárgyalom.

## Rendszer

### Beállítás

Itt két jellemzőjét tudjuk beállítani a programunknak, a stílust és a háttér színét, ami végigkísér bármely funkció használatakor. Kattintsunk is rá erre a két jellemzőre és nézzük meg, mik közül választhatunk. A stílusnál Classic, Xp és Metal kinézetek között választhatunk, amik a Swing beépített stílusai is egyben. A szín kiválasztását többféleképpen megadhatjuk. Az első fülön a felkínált színekből, a másodikon és harmadikon, HSB és RGB kódokkal megadott színekből tudunk választani. A színeknél még van egy alapértelmezés gomb is, amely az első induláskor látott színt állítja be. Fontos, hogy a változtatások csak újra bejelentkezés után fognak ténylegesen megváltozni.

### Log fájl

A rendszer részletesen naplózza az egyes bejelentkezett felhasználók által elvégzett műveleteket. Így hiba esetén vagy csak tájékozódás céljából vissza tudjuk keresni a kívánt információkat. A program naponta logol, azaz minden nap készít egy külön napló fájlt, ami az aznapi cselekményeket tartalmazza. A Log fájl menüpont megnyitásakor, az aktuális napra vonatkozó események jelennek meg. Alul a beviteli mezőbe szabályos dátumot beírva, ugorhatunk az egyes fájlok között. Üres mező esetén, mindig az aznapi adatok köszönnek vissza. A fájl szerkezete tartalmazza a pontos dátumot, az információ szintjét (Például INFO az információs, ERROR a hibaüzenetekhez), azt hogy melyik osztályban történt a kiváltó cselekmény és egy értelmes üzenetet. Tetszés szerint ki is nyomtathatjuk az adatokat. A Log fájlok tárolása a Fájlok/log_file könyvtárban kapott helyett.

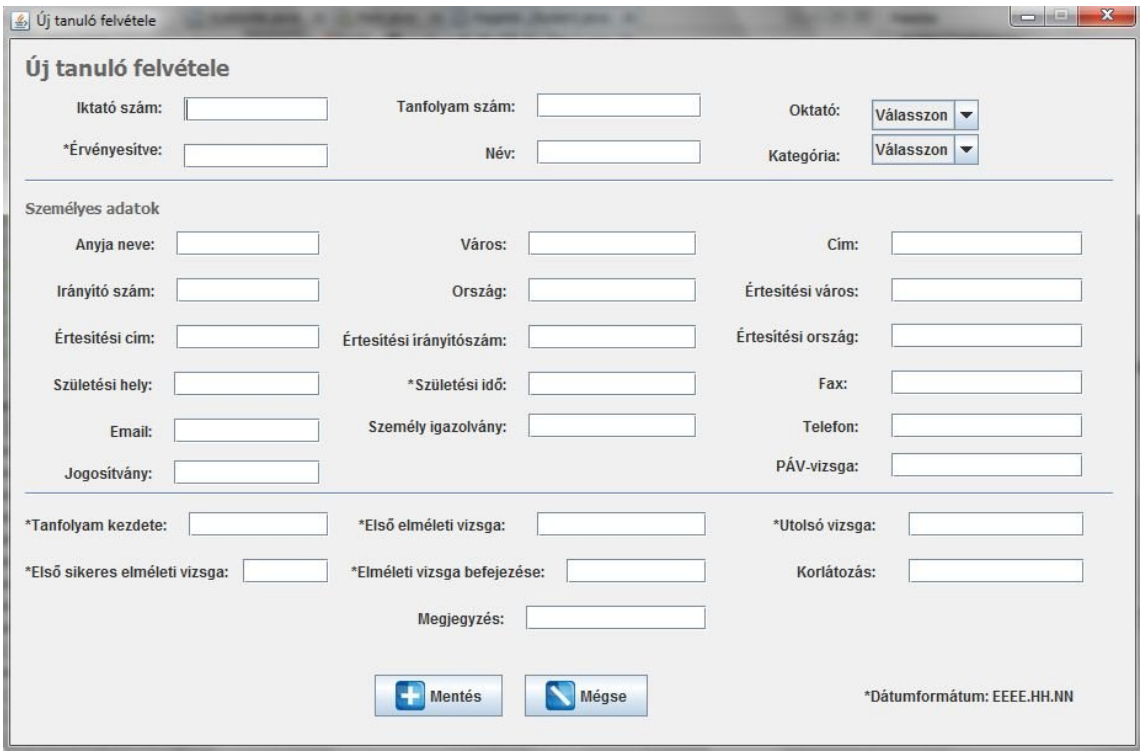
Továbbá két féle kilépés létezik a főmenüből. Az első a kilépés és bejelentkezés, amivel a bejelentkező panelre ugorhatunk más felhasználóval való bejelentkezés, szín, vagy stílusváltoztatás miatt. A második kilépés funkcióval végérvényesen kilépünk és bezáródik az alkalmazás.

## Műveletek

Ez a törzsmenü lesz a leghasznosabb és leginkább használt része a programnak. Itt tudunk új tanulókat, oktatókat, gyakorlati oktatásokat létrehozni és lekérdezéseket megvalósítani, vagy nyomtatványokat szerkeszteni.

### Új tanuló

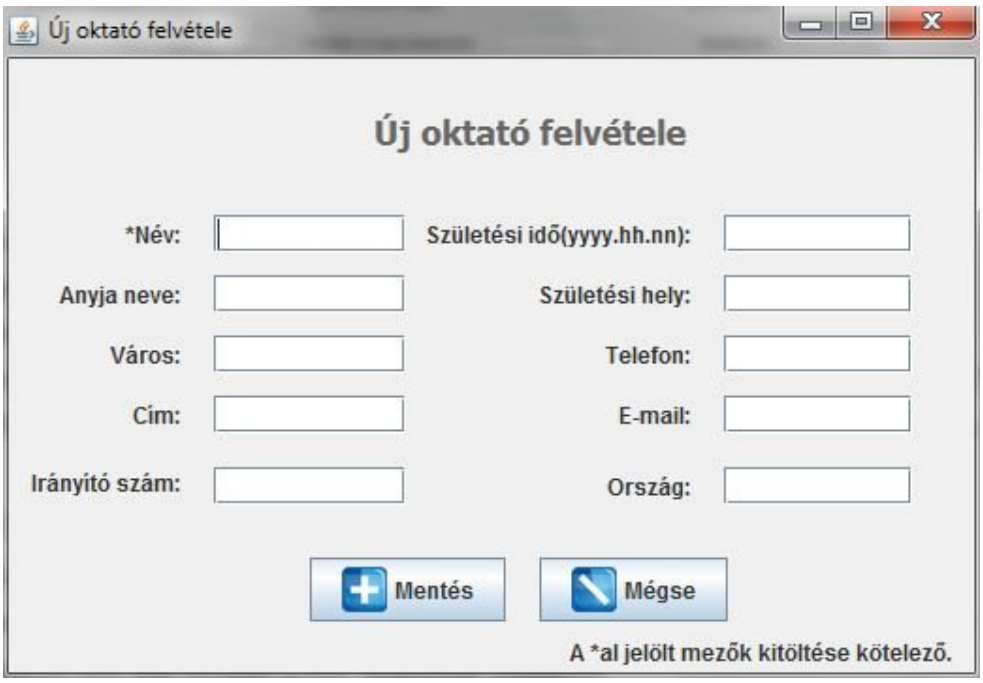
Mint a neve is elárulja új tanulók felvételére alkalmas.




A felső részben lévő adatok kitöltése kötelező, ezért ha még nincs oktató regisztrálva a programban, akkor a tanulót nem tudjuk elmenteni. Az * - gal jelölt mezők kötelezően kitöltendők, szabványos, jobb - lent leírt alakban, amit egy külön Java osztály ellenőriz. A választható kategóriák: A1, Akorl, A, B, C1, C, D1, D, B+E, C1+E, C+E, D1+E, D+E, M, T, TR. Mint látható ezen kívül is sok adat menthető, melyek már a felhasználóra vannak bízva.

### Új oktató

Egy autósiskola legfontosabb „alkatrészei”. Rájuk nem vonatkozik annyi adat, mint a tanulókra. Értelemszerűen kell kitölteni itt is a mezőket, kötelezően csak a név szerepel, de ajánlott több adat felvitele, a jobb tájékozódás érdekében.



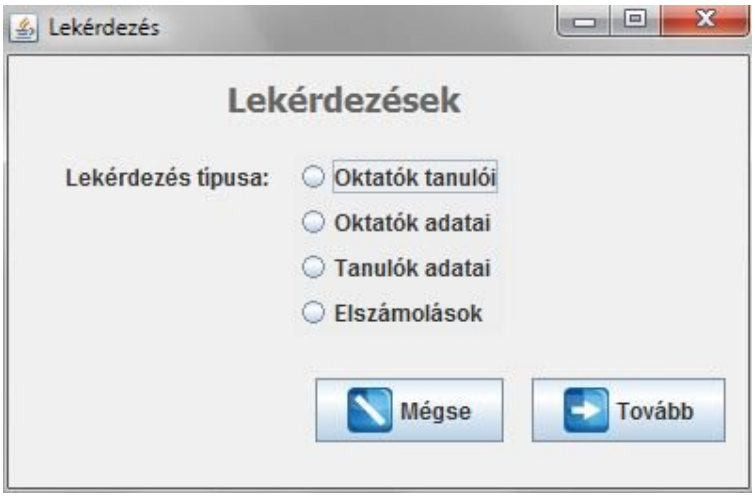

Mind a tanulók és mind az oktatók esetében, ugyanazon névvel új embert felvenni nem lehet és ilyenkor hiba üzenetet kapunk, hogy az illető már létezik.

### Új oktatás hozzáadása

Ha már vannak tanulóink, oktatóink akkor az együtt eltöltött gyakorlati oktatást is fel kell jegyeznünk. Ezt a célt szolgálja ez a menüpont. Felülről lefele haladva először az oktatót kell kiválasztanunk, így a tanulók listájában rögtön megjelennek, hogy kik tartoznak az adott oktatóhoz. Végül az oktatás típusát se felejtsük el megadni. Lejjebb már a konkrét adatokat kell megadnunk és végül a mentés gomb. Rögtön feltűnik a táblázatunkban az új elem, ha nem rontottunk el semmit. Törölni, ebből a táblázatból kijelölve egy sort és azután a törlés gomb megnyomásával lehet. Itt is nyomtathatunk a jobb átláthatóság kedvéért. Ha már több elem is van a táblázatunkban, akkor megfigyelhetjük, hogy a panelen alul összesítődnek a levezetett kiló méterek és órák.

**Lekérdezések**

A különféle információk megjelenítésére, illetve nyomtatására való menüpont.

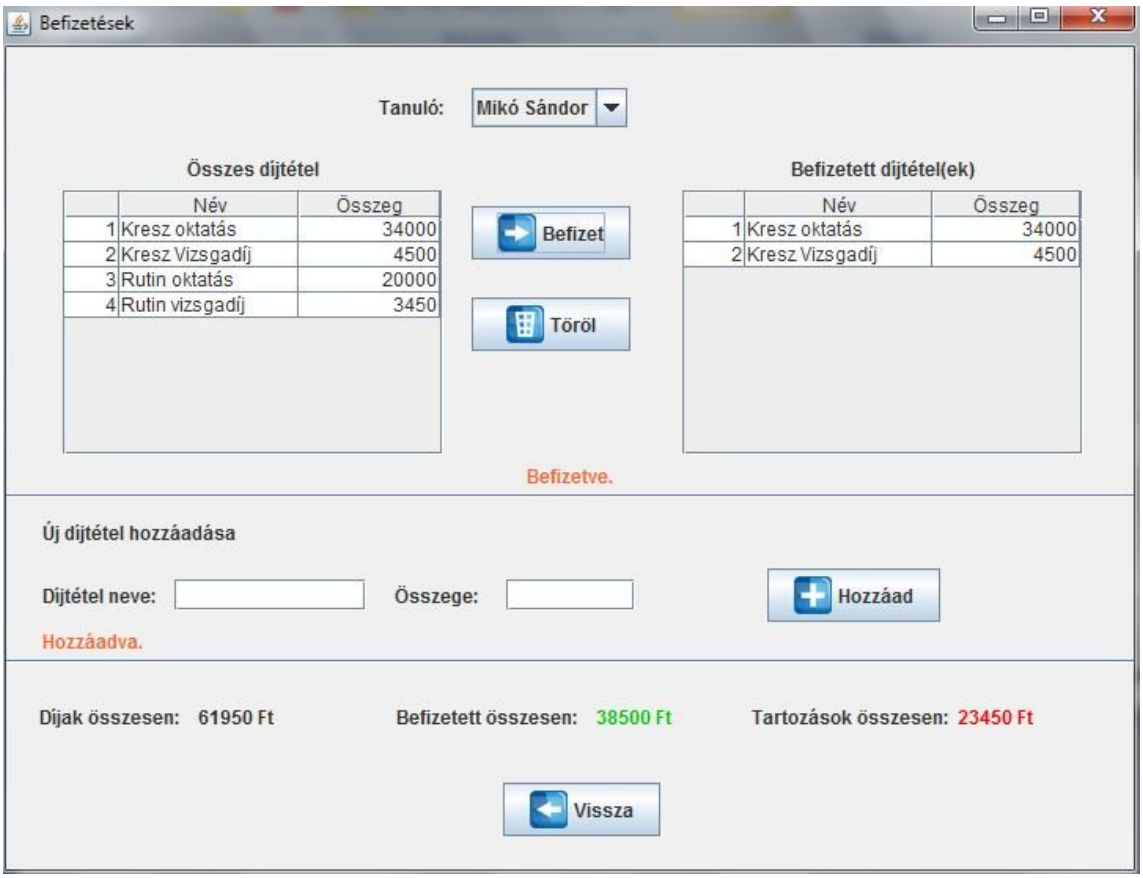



Az oktatók és tanulók adatai, felsorolja az összes hozzátartozó személyt, fontosabb jellemzőivel. Az oktatók tanulóinál, az általunk kiválasztott oktatóhoz tartozó összes tanulót és adatait kérdezhetjük le.

Ezek után már csak egy menüpont maradt, aminek a bemutatását külön tárgyalnám.

**Elszámolások**

Az egyes tanulókhoz itt tudjuk nyílván tartani és szerkeszteni, az anyagi elszámolásokat. Először kiválasztjuk a tanulót, ilyenkor megjelenik a hozzátartozó összes és befizetett díjtétel. Új tétel hozzáadása panelrészben, adhatunk újakat hozzá az összes díjtételhez. Ahhoz, hogy ezek befizetetté váljanak, kijelölés után a befizet gombbal válik lehetségessé és ilyenkor átkerül az aktuális tétel, a befizetett részhez. Törölni szintén az összes megjelenített közül tudunk, hasonlóan, a törlés gombbal. Ekkor eltűnik, ha szerepelt, mindkét listából. Még egy hasznos információt kapunk ebből a menüpontból, mégpedig a lejjebb, különböző színekkel feltűntetett aktuális állásról. Megtudhatjuk, hogy mennyi a tanulóhoz összesen kiírt, befizetett, tartozás díjtétel. Ezek dinamikusan változnak, ahogy a felhasználó manipulálja őket.




**Nyomtatványok**

Ezen belül találjuk a szerződést, mint előre megszerkesztett dokumentumot. Fontos, hogy a program automatikusan kitölti a cég adatokat a szerződésben, ha azok meg vannak adva. Ezért ha még nem tettük meg, a Szerkesztés/Cég adat szerkesztése menüpont alatt lehetőségünk van rá, vagy a szerződésben mi magunk is módosíthatjuk. A szerződés többi szöveges részei is manipulálhatóak a nyomtatás előtt, ha nem találunk valamit ideálisnak.

## Szerkesztés

Ez a menüpont foglalkozik az adatok módosításával és törlésével.

A cég adatainak szerkesztésénél megadhatjuk a szerződéshez is szükséges törzsadatokat. Az oktató, illetve tanuló adatváltoztatásnál, a nevek kiválasztása után betöltődnek a már régebben megadott adatok, és ezek átírásával, kitörlésével megváltoztathatjuk azokat.

Sőt ha egyszerűen az egész személyt akarjuk kitörölni a rendszerünkből, akkor a törlés gomb lesz a jó választás.

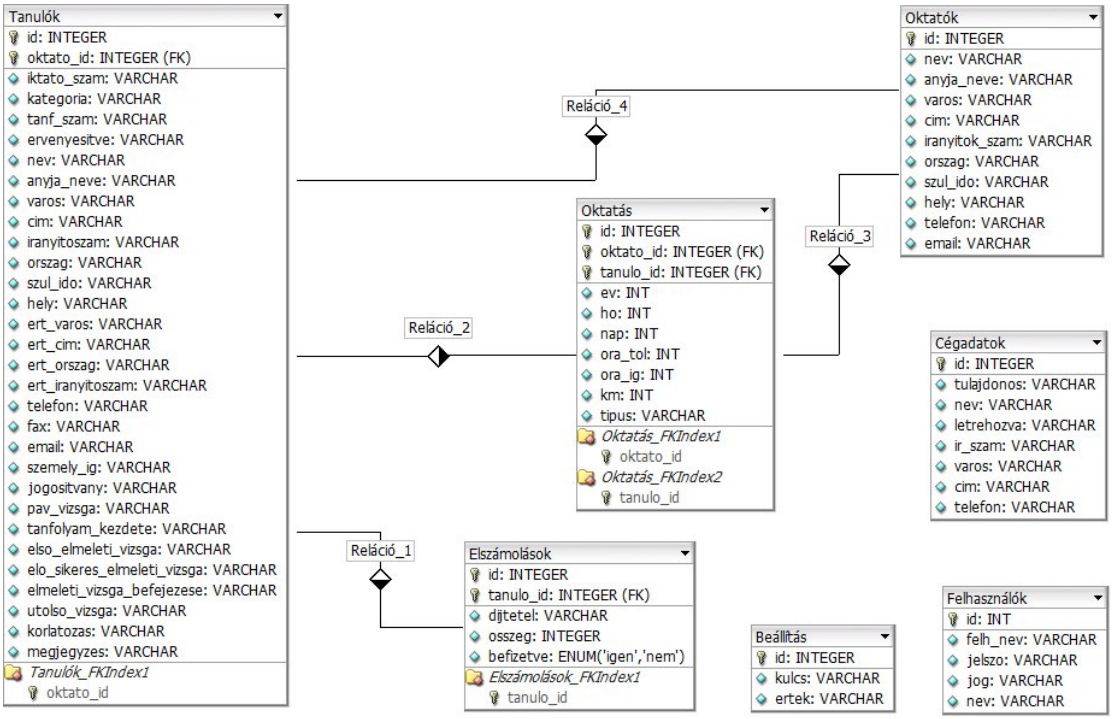
## Egyéb

Ide kerültek az egyéb, kiegészítő lehetőségek. Ezek többek között hasznos eszközök, információk.

Számológép a Windows beépített eszközét nyitja meg. Jegyzettömb esetében is a Windowsban található Notepad köszön vissza. Végül a névjegy információkat tartalmaz a programmal kapcsolatban.

## Adatbázis

A legtöbb eddig bemutatott funkció használ adatbázis elérést. Azért, hogy az adatok átláthatóak és rendezettek legyenek, különböző adatbázis tábláknak kell létezniük a háttérben. A program 7 táblát használ, amiknek a kapcsolatát a következő ábra mutatja:



## 4.3. A program gyakorlati jelentősége

A program gyakorlati jelentősége abban rejlik, hogy egyszerű, átlátható és mindenki számára könnyen érthető. Célomat, hogy egy egyszerű, felhasználóbarát nyilvántartó programot készítsek, sikerült teljesítenem. Miután elkészítettem a programot, fontosnak tartottam, hogy az autósiskolát vezető ismerősömnek és egy olyan embernek is megmutassam, aki kevésbé ért az informatikához, a programozáshoz.

Az autósiskolát vezető ismerősömnek, aki már több ilyen nyilvántartó rendszerrel találkozott, tetszett az elkészült program. Kipróbálta a programot, összehasonlította a most használt programjukkal, és jelentősen hatékonyabbnak, eredményesebbnek tartotta. Kiemelte, hogy milyen egyszerű és könnyű a használata, a mostani rendszerükkel ellentétben.

Fontosnak tartottam azonban, hogy egy kívülálló véleményét is kikérjem, aki nem járatos a programok készítésében és tényleg csak felhasználói szinten használja a számítógépet és az internetet is. Elmondása szerint, könnyen kezelhető a program, az utasítások, parancsok egyértelműek, érthetőek, és alkalmas a program az adatok nyilvántartására.

# 5. ÖSSZEGZÉS

Szakdolgozatomban azt vizsgálom, hogy a Java Swinggel hogyan lehet egyszerű és egyben gyors felhasználói felületet készíteni és hogy ez alkalmas - e egy komolyabb, gyakran használt, üzleti logikájú program megvalósításához.

A program elkészítésében az volt a célom, hogy egy átláthatóbb, egyszerűbb tanuló nyilvántartó rendszert tervezzek egy autósiskola számára. A program megvalósításában a fő hangsúly azon volt, hogy egy olyan rendszert hozzak létre, amely még inkább személyre szabott és nem tartalmaz felesleges extra funkciókat, ezzel is elősegítve a hatékonyságot.

Célomat sikerült véghezvinni, az elkészült program egyszerű, könnyen kezelhető, érthető, nem tartalmaz felesleges funkciókat, a lényegre koncentrál, elősegíti, a felhasználó kényelmét és az adatok hatékony nyilvántartását, rendszerezését. Programom előnyeit nemcsak az autósiskola vezetője ismerte el, hanem egy hétköznapi, a témában nem jártas ember is.

# 6. IRODALOM

Angster Erzsébet: Objektumorientált terezés és programozás 1. kötet. Martonvásár: Akadémiai nyomda, 2001

Angster Erzsébet: Objektumorientált terezés és programozás 2. kötet. Martonvásár: Akadémiai nyomda, 2002

Cornelia Heinisch, Frank Müller-Hofmann, Joachim Goll: Java als erste Programmiersprache, Wiesbaden: Teubner Verlag, 2000

Juhász Istvá: Magas szintő programozási nyelvek 2. Egyetemi jegyzet. Debrecen: mobiDIÁK könyvtár, 2009

http://www.apachefriends.org/en/xampp-windows.html

http://java.sun.com/j2se/1.4.2/docs/api/javax/swing/package-summary.html

http://www.java.com/en/download/index.jsp

http://netbeans.org